Cpminer Manual

download_gb_file.py

1. Function

This script downloads chloroplast genome files in **.gb** format in batches from the NCBI GenBank database. It supports multithreading and batch requests to optimize download efficiency while adhering to NCBI API request limits (not more than 3 requests per second).

2. Prerequisites

1. Install the required libraries:： pip install biopython pandas func-timeout
2. Register your email with NCBI (replace **your_email@example.com** in the script).

Parameter description

Parameter	Abbreviation	Required	Default	Description
--your_email	-e	No	your_email@example.com	Your email (e.g., your_email@example.com) for NCBI API usage
--input_file	-i	Yes	-	Input file (CSV/XLSX format), containing accession IDs
--output_dir	-o	Yes	-	Output directory
--threads	-t	No	10	Number of threads
--batch_size	-b	No	3	Batch size for accession processing

3. Input File Format

- **File Type:** CSV or Excel (.xlsx)
- **Required Column:** accession (must contain GenBank accession numbers, e.g., NC_000001.1)
- **Example Content:**

CSV

```
accession
NC_000001.1
NC_000002.1
```

4. Usage Examples

bash

```
# Basic usage
python download_gb_file.py -i accessions.csv -o ./gb_files -e your_email@example.com

# Using 20 threads and 5 accessions per batch
python download_gb_file.py -i data.xlsx -o ./genomes -t 20 -b 5
```

5. Workflow

1. Read Input: Parses the list of accessions from the CSV/XLSX file.
2. Filter Duplicates: Skips existing .gb files in the output directory.
3. Task Assignment:
    - Splits accessions into batches (3 per batch by default).
    - Processes batches concurrently using multiple threads (10 threads by default).
4. Rate Limiting: Automatically limits request frequency (to ge0.34 seconds per request).
5. Download Retries: Automatically retries failed downloads (up to 3 times).
6. Save Results: Saves files named after the base accession (e.g., NC_000001.gb).

6. Output

- **Directory Structure:**

  bash

```
output_dir/
├── NC_000001.gb
├── NC_000002.gb
└── ...
```

- **File Naming:** The file name removes the accession version number (e.g., NC_000001.11 becomes NC_000001.gb).

7. Important Notes

1. **API Limitations:**

- A single request can support a maximum of 3 accessions (due to NCBI policy).
- High concurrency may lead to a temporary IP ban; it's recommended to use le20 threads.

2. **Timeout:** The timeout for a single batch download is 10 minutes.
3. **Error Handling:**

- Failed downloads are automatically retried 3 times.
- Any accessions that ultimately fail to download will be reported in the terminal.

quality_ctrl.py

1. Function

This script performs quality control and re-annotation of chloroplast genome files in .gb format. It primarily has two main functions:

1. **Quality Filtering:** Analyzes genome features and filters out files that do not meet quality standards.

2. **Re-annotation:** Uses PGA (Plastid Genome Annotator) to re-annotate problematic files.

2. Prerequisites

1. **Install the required libraries:**

2. **pip install biopython pandas**

3. **Install the necessary tools:**

- Perl (v5.10+)
- BLAST+ (v2.6+)
- PGA.pl (must be downloaded separately)

3. Parameter description

Parameter	Abbreviation	Required	Default	Description
--input_dir	-i	Yes	-	Input directory (containing .gb files)
--output_dir	-o	Yes	-	Output directory (saving problematic files and results)
--cds_threshold	-c	No	80	Minimum CDS count threshold
--ambig_threshold	-a	No	0.2	Maximum ambiguous base proportion threshold
--PGA_path	-PGA	No	-	Path to the PGA.pl script
--inverted_repeat	-ir	No	1000	Minimum inverted repeat region length
--pidentity	-p	No	80	BLAST identity percentage threshold
--reference	-r	No	-	Reference genome directory (closely related species)
--threads	-t	No	3	Number of processing threads

4. Quality Control Standards

1. **CDS Count:** A file is flagged if its number of CDS is below a set threshold (default: <80).

2. **Unclear Base Ratio:** A file is flagged if the percentage of ambiguous bases is above a set threshold (default: >20%).

3. **Handling Problematic Files:**

- The file is converted to FASTA format and saved to the output directory.
- The original GB file is deleted.
- A quality report, gb_info.csv, is generated.

5. Re-annotation Workflow

1. **Dependency Check:** Automatically detects the paths for Perl and BLAST.

2. **File Grouping:** Files are grouped according to the number of threads for processing.

3. **PGA Annotation:** The PGA.pl script uses reference genomes for re-annotation.

Perl

```
perl PGA.pl -r [reference_directory] -i [input_file] -o [output_directory]
```

1. **Result Integration:** Merges the annotated results and cleans up temporary files.

6. Output Files

1. **Quality Report (gb_info.csv):**

csv

```
filename,organism,sequence_length,gene_count,CDS,tRNA,rRNA,unclear_bases,unclear_ratio,error
```

2. **Problematic Files:** Converted to FASTA format (.fasta).

3. **Re-annotation Results:** New GB files (.gb).

7. Usage Examples

Bash

```
# Basic quality filtering
python quality_ctrl.py -i ./genomes -o ./qc_output -c 70 -a 0.15
 -t 8

# Quality filtering + re-annotation
python quality_ctrl.py -i ./genomes -o ./qc_output \
    -PGA ./tools/PGA.pl \
    -r ./reference_genomes \
    -ir 1500 -p 85 -t 10
```

8. Important Notes

1. **Re-annotation Dependencies:**
    - You must provide the PGA_path and reference parameters.
    - The reference genomes should be from a closely related species within the same family.
2. **Path Requirements:**
    - The input directory must contain .gb files.
    - The PGA.pl path must point to an executable script.
3. **Recommended Thresholds:**
    - A complete chloroplast genome typically has >80 CDS.
    - High-quality genomes usually have <5% ambiguous bases.
4. **Resource Consumption:**
    - Each thread requires approximately 1 GB of memory.
    - A high-performance server is recommended for large datasets.

get_cds.py

1. Function Description

This script extracts coding sequences (CDS) and ribosomal RNA (rRNA) genes from chloroplast genome files in .gb format. Its main features include:

1. **Gene Extraction:** Extracts 85 standard chloroplast gene sequences from GB files.
2. **Sequence Standardization:** Unifies gene naming conventions and saves the sequences in FASTA format.
3. **Result Statistics:** Generates a statistical report on gene counts and locations.

2. Core Features

- Supports the extraction of 85 standard chloroplast genes (including CDS and rRNA).
- Automatically handles differences in gene naming (e.g., "16S ribosomal RNA" is standardized to "rrn16").
- Uses multi-threading for parallel processing to improve efficiency.
- Automatically cleans up empty result files.

3. Parameter Description

Parameter	Abbreviation	Required	Default	Description
--input_dir	-i	Yes	-	Input directory (containing .gb files)
--output_dir	-o	Yes	-	Output directory (saving gene sequences)
--threads	-t	No	3	Number of processing threads

4. The script supports extracting the following 85 standard chloroplast genes:

Python

```
['accD', 'atpA', 'atpB', 'atpE', 'atpF', 'atpH', 'atpI', 'ccsA', 'cemA',
```

```
 'clpP', 'infA', 'matK', 'ndhA', 'ndhB', 'ndhC', 'ndhD', 'ndhE', 'n
dhF', 'ndhG', 'ndhH', 'ndhI',
 'ndhJ', 'ndhK', 'petA', 'petB', 'petD', 'petG', 'petL', 'petN', 'p
saA', 'psaB', 'psaC', 'psaI',
 'psaJ', 'psbA', 'psbB', 'psbC', 'psbD', 'psbE', 'psbF', 'psbH', 'p
sbI', 'psbJ', 'psbK', 'psbL',
 'psbM', 'psbN', 'psbT', 'psbZ', 'rbcL', 'rpl14', 'rpl16', 'rpl2',
'rpl20', 'rpl22', 'rpl23',
 'rpl32', 'rpl33', 'rpl36', 'rpoA', 'rpoB', 'rpoC1', 'rpoC2', 'rps1
1', 'rps12', 'rps14', 'rps15',
 'rps16', 'rps18', 'rps19', 'rps2', 'rps3', 'rps4', 'rps7', 'rps8',
 'rrn16', 'rrn23', 'rrn4.5',
 'rrn5', 'ycf1', 'ycf15', 'ycf2', 'ycf3', 'ycf4', 'ycf68']
```

5. Output Files

1. Gene Sequence Files:

- **Each gene is saved individually as a .fasta file (e.g., rbcL.fasta).**
- **Sequence header format: >{Genome_ID}:{Gene_Location}**

2. Statistical Reports:

- **cds_num.csv: Statistical count of each gene within each genome.**

csv

```
accession,organism,accD,atpA,...,cds_num
```

- **cds_loc.txt: Detailed location information for each gene.**

Plaintext

```
accession   accD   atpA   ...    cds_num
```

6. Usage Examples

Bash

```
# Basic usage
python get_cds.py -i ./genomes -o ./cds_sequences

# Speed up with 8 threads
python get_cds.py -i ./gb_files -o ./gene_seqs -t 8
```

---

7. Processing Workflow

1. **Initialization:**
    - **Creates the output directory.**
    - **Creates empty FASTA files for each gene.**
2. **Multi-threaded Processing:**
    - **Each thread processes one GB file.**
    - **Extracts gene sequences and appends them to the corresponding gene file.**
    - **Records gene counts and location information.**
3. **Result Integration:**
    - **Merges statistical results from all threads.**
    - **Generates CSV and TXT report files.**
4. **Cleanup:**
    - **Deletes empty result files.**

8. Important Notes

1. **Input Requirements:**
    - **The input directory must contain .gb files.**
    - **It's recommended to run the quality filtering script first to ensure file integrity.**
2. **Thread Settings:**

- **The default is 3 threads; adjust based on your CPU core count.**
- **More threads (8-16) are recommended for large datasets.**

3. **Sequence Processing:**
    - **Automatically handles reverse complementary strands.**
    - **Merges multi-segment genes (e.g., genes separated by introns).**
4. **Gene Naming:**
    - **Supports automatic conversion of common aliases.**
    - **Unrecognized genes are marked as "unknown."**

filter_seq.py

1. Function Description

This script filters coding sequences (CDS) based on length. It compares the lengths of input sequences to reference gene lengths (for either angiosperms or gymnosperms) and retains only those within a specified range. Its main features include:

1. **Length Filtering:** Keeps only sequences whose lengths fall within a defined multiple of the reference length.
2. **Sequence Selection:** When multiple sequences for the same gene appear in a single genome, only the longest sequence is retained.
3. **Result Statistics:** Generates a statistical report containing the lengths of each gene.

2. Core Features

1. Filters sequences based on reference gene lengths for either angiosperms or gymnosperms.
2. Automatically handles multi-copy genes by keeping the longest copy.
3. Uses multi-threading for parallel processing to improve efficiency.
4. Generates a detailed length statistics report.

3. Parmeter Description

Parameter	Abbreviation	Required	Default	Description
--input_dir	-i	Yes	-	Input directory (contains .fasta files)

--output_dir	-o	Yes	-	Output directory (saves filtered sequences)
--ref_orthologous	-r	Yes	-	Reference type: Ang (angiosperms), Gym (gymnosperms)
--lower_bound	-lb	No	0.5	Lower length bound (multiple of reference length)
--upper_bound	-ub	No	2.0	Upper length bound (multiple of reference length)
--threads	-t	No	3	Number of processing threads

4. Filtering Rules

1. **Length Range:** A sequence is kept only if its length is within the range of Reference_Length × Lower_Bound ≤ Sequence_Length ≤ Reference_Length × Upper_Bound.
2. **Handling Duplicate Sequences:**
    - If multiple sequences for the same gene appear in a single genome,
    - Only the longest sequence is considered for length filtering.

5. Reference Gene Lengths

The script includes standard gene lengths for two plant categories:

- **Angiosperms:**

Python

```
{'accD':1599, 'atpA':1524, ... 'ycf68':234}
```

- **Gymnosperms:**

Python

```
{'accD':1041, 'atpA':1524, ... 'ycf68':240}
```

6. Output Files

- **Filtered Sequences:**
    - Each gene is saved as a separate **FASTA** file (e.g., rbcL.fasta).
    - The sequence header format remains unchanged.
- **Statistical Report (length.csv):**

```
csv
accession,organism,accD,atpA,...
```

    - Records the final sequence length for each gene in each genome.
    - Genes that fail the filtering process are left blank.

7. Usage Examples

```
Bash
# Angiosperm reference, default range (0.5-2.0)
python filter_seq.py -i ./cds -o ./filtered_cds -r Ang

# Gymnosperm reference, custom range (0.6-1.8)
python filter_seq.py -i ./gene_seqs -o ./filtered_genes -r Gym
-lb 0.6 -ub 1.8 -t 8
```

8. Important Notes

1. **Input Requirements:**
    - The input directory must contain cds_num.csv (generated by get_cds.py).
    - **FASTA** filenames must match the gene names in the reference dictionary (e.g., rbcL.fasta).
2. **Parameter Ranges:**
    - The lower bound is recommended to be ge0.5 and the upper bound le2.0 (can be adjusted).
    - The number of threads should be set based on your CPU core count.
3. **Special Handling:**
    - Unknown genes (not in the reference dictionary) will be skipped.
    - Empty result files are automatically ignored.

select_seq_by_acc.py

1. Function Description

This script selects a representative chloroplast genome for each species. Its primary functions are:

1. Standardizing Species Names: Corrects scientific names using a provided species name lookup table.
2. Selecting Representative Genomes: For species with multiple genomes, it chooses the one with the longest total CDS length as the representative.
3. Renaming Sequences: Adds the standardized species name to the sequence description.

2. Core Features

1. Automatically detects and handles multiple genomes for the same species.
2. Supports an external species name lookup table for correcting scientific names.
3. Uses multithreading to process gene files in parallel.
4. Generates a table that maps each species to its representative genome.

3. Parameter Description

Parameter	**Abbreviation**	**Required**	**Default**	**Description**
--input_dir	-i	Yes	-	Input directory (contains filtered CDS files)
--output_dir	-o	Yes	-	Output directory (saves representative genome sequences)
--file_organism_name	-f	No	-	Species name mapping table (CSV format)
--threads	-t	No	3	Number of processing threads

4. Processing Workflow

1. **Data Preparation:**
    - The script reads the length.csv file, which is generated by a previous script.
    - It then calculates the total CDS length for each genome listed in the file.
2. **Species Name Standardization:**
    - If a lookup table is provided, the script uses it to correct and standardize scientific names.
    - It replaces any spaces in the species names with underscores (e.g., Arabidopsis thaliana becomes Arabidopsis_thaliana).
3. **Representative Genome Selection:**
    - Genomes are grouped by species.
    - The script selects the genome with the longest total CDS length as the **representative** for that species.
4. **Sequence Extraction:**
    - Using multiple threads, the script processes each gene file.
    - It extracts and saves only the sequences from the representative genome for each species.
    - The standardized species name is added to the header of each sequence.

5. Special Handling Mechanisms

1. **Handling Duplicate Genomes:** If the script finds multiple genomes for a single species, it will interactively prompt the user, asking whether to keep the genome with the longest total CDS length. The user can either input y to continue or n to terminate the program.
2. **Species Name Standardization:** The script prioritizes names from the provided lookup table. If a species is not found in the table, its original name is retained.

6. Input/Output Examples

- **Input File Structure:**

```
Bash
input_dir/
├── atpA.fasta
├── rbcL.fasta
├── ...
└── length.csv  # Required file
```

- **Output File Structure:**

```
Bash
output_dir/
├── atpA.fasta     # Contains only representative genome sequen
ces
├── rbcL.fasta
├── ...
└── organism_nuique.csv  # Species-genome mapping table
```

- **Species Name Lookup Table Example (file_organism_name.csv):**

```
csv
```

```
accession,organism
NC_000932.1,Arabidopsis thaliana
NC_037304.1,Oryza sativa
```

- **Sequence Header Format Change:**

```
Diff
- >NC_000932.1:123..456
+ >NC_000932.1:123..456|Arabidopsis_thaliana
```

7. Usage Examples

```
Bash
# Basic usage (without species name correction)
python select_seq_by_acc.py -i ./filtered_cds -o ./representati
ve_seqs

# Using a species name lookup table with 8 threads
python select_seq_by_acc.py -i ./genes -o ./species_reps \
    -f species_names.csv -t 8
```

8. Output File Descriptions

- **Gene Sequence Files:**
  - The original filenames (e.g., rbcL.fasta) are kept.
  - The files contain only the representative sequences for each species.
  - The standardized species name is added to each sequence header.
- **Species Table (organism_nuique.csv):**

```
csv
accession,organism,cds_num,length
NC_000932.1,Arabidopsis_thaliana,85,12345
```

  - **accession:** The genome accession number.
  - **organism:** The standardized species name.
  - **cds_num:** The number of genes.
  - **length:** The total CDS length.

9. Important Notes

- **Required Input Files:** The input directory must contain length.csv (generated by the filter_seq.py script) and gene files in FASTA format (.fa, .fas, .fasta).
- **Species Name Lookup Table:** The table must have the columns accession and organism. Any species not included in the table will retain its original name.
- **Thread Settings:** The default is 3 threads, but you can increase this number for larger datasets. Each thread processes one gene file.

cds_align.py

1. Function Description

This script uses MAFFT to perform multiple sequence alignment on coding sequences (CDS). Its main features include:

- **Multiple Sequence Alignment:** Accurately aligns homologous sequences for each gene.
- **Parallel Processing:** Supports multi-process execution to handle multiple genes simultaneously.
- **Command Validation:** Automatically validates the user-provided MAFFT command format.
- **Path Detection:** Intelligently detects the MAFFT installation location on your system.

2. Prerequisites

- **MAFFT Installation is Required:**
    - **Linux/macOS: Use sudo apt install mafft or brew install mafft.**
    - **Windows: Download a pre-compiled version and add it to your system's PATH.**
- **Input File Requirements:**
    - **Sequence files must be in FASTA format (.fasta, .fas, .fa).**
    - **Each file must contain sequences of the same gene from different species.**

3. Parameter description

Parameter	Abbreviation	Required	Default	Description
--input_dir	-i	Yes	-	Input directory (containing CDS files)
--output_dir	-o	Yes	-	Output directory (Save the comparison results)
--para_file	-p	No	3	The number of files processed in parallel

4. MAFFT Command Format Requirements

You must provide a MAFFT command template that includes the following placeholders:

1. **in.fasta:** Placeholder for the input file.
2. **out.fasta:** Placeholder for the output file.
3. **Standard Format:** mafft [options] in.fasta > out.fasta

```
Examples of Valid Commands:
Bash
mafft --auto --thread 4 --reorder in.fasta > out.fasta
mafft --localpair --maxiterate 1000 in.fasta > out.fasta
```

5. Workflow

1. **Path Detection:** The script automatically looks for MAFFT in your system's PATH. If not found, it prompts you to enter the path manually.
2. **Command Validation:** It checks if the command format is correct and verifies if the options are supported by MAFFT.
3. **Task Assignment:** Alignment tasks are created for each gene file.
4. **Parallel Alignment:** Multiple processes are used to align genes concurrently. The script displays real-time progress and error messages.
5. **Result Saving:** The alignment results are saved in **FASTA** files with the same name as the input files, preserving the original sequence headers.

6. Usage Examples

```
Bash
# Basic usage
python cds_align.py -i ./cds_sequences -o ./aligned_cds

# Advanced usage (8 processes + custom parameters)
python cds_align.py -i ./genes -o ./aligned_genes -p 8
```

Example of Interactive Steps:

- The script either automatically detects the MAFFT path or prompts for a manual input.
- You enter the MAFFT command template:

Bash

mafft --auto --thread 4 --reorder in.fasta > out.fasta

- The script validates the command and proceeds with the alignment.

---

7. Output

Output Directory Structure:

```
Bash
output_dir/
 ├── atpA.fasta       # Aligned ATP synthase gene
```

```
├── rbcL.fasta       # Aligned Rubisco large subunit gene
└── ...              # Other gene alignment results
```

Sequence Header Format Remains Unchanged:

```
csv
>NC_066001:[24468:25851](-)|Achillea_ageratum
ATGGT...TGACA
```

8. Performance Recommendations

- **Thread Settings:**
  - Small datasets (<50 genes): The default of 3 processes is sufficient.
  - Large datasets (>100 genes): Use 8-16 processes.
- **MAFFT Parameter Selection:**
  - For fast alignment, use --auto.
  - For high-accuracy alignment, use --localpair --maxiterate 1000.
- **Memory Management:**
  - Each process requires approximately 500 MB to 1 GB of memory.
  - Large genes (e.g., ycf1) may require more memory.

9. Important Notes

- **File Naming:** Input files must be named by gene (e.g., rbcL.fasta) and cannot contain spaces.
- **Command Limitations:** The command must include in.fasta and out.fasta as placeholders and must use > for output redirection.
- **Error Handling:** The script will prompt you to re-enter the command if it's invalid. Any failed alignments will display an error message.

trim_start_end3.py

1. Function Description

This script performs a two-step trimming process on multiple sequence alignment files:

- **Boundary Trimming:** Removes unaligned regions from the ends of sequences where the gap percentage exceeds a specific threshold.
- **Internal Trimming (Optional):** Uses the trimAL tool to remove unaligned fragments from the interior of the sequences.

2. Prerequisites

- **Python Environment:**
  - **Python 3.x is required.**
  - **Install the necessary libraries: biopython and argparse.**
  - **Installation command: pip install biopython argparse**
- **External Dependency (Required for Internal Trimming Only):**
  - **The trimAL tool (available on its official GitHub).**
  - **You must specify the path to the executable file using the -tp parameter (e.g., /usr/bin/trimal).**

3. Parameter Descrption

Parameter	Abbreviation	Required	Default	Description
--input_dir	-i	Yes	-	Input directory (stores FASTA format alignment files)
--output_dir	-o	Yes	-	Output directory (saves trimming results)
--boundaries_threshold	-bt	No	0.025	Boundary gap ratio threshold (between 0-1)

--trimal_path	-tp	No	-	Path to the trimAL executable file (enables internal trimming)
--input_dir	-i	Yes	-	Input directory (stores FASTA format alignment files)

4. Usage Examples

```
Bash
# Basic usage (boundary trimming only)
python trim_start_end3.py -i input_fasta_dir -o trimmed_results -bt 0.03

# Full process (boundary + internal trimming)
python trim_start_end3.py -i input_dir -o output_dir -tp /opt/trimal/bin/
trimal -p 4

# Using 8 parallel processes
python trim_start_end3.py -i alignments -o results -p 8
```

5. Processing Workflow

1. **Boundary Trimming (mandatory for all files):** The script scans from both the start and end of the sequences, removing columns where the gap percentage exceeds a user-defined threshold. It then outputs the remaining core sequence region.

2. **Internal Trimming (triggered with -tp):** If the path to the trimAL executable is provided, the script uses trimAL's -automated1 mode to automatically detect and remove internal, unaligned regions. This process overwrites the output from the boundary trimming step.

6. Output Details

1. Output filenames remain the same as the input files (e.g., gene1.fasta becomes gene1.fasta).

2. All output files are saved to the directory specified by the --output_dir parameter.

7. Performance Tips

1. **Parallel Processing:** Adjust the number of parallel files to process using the -p parameter. It is recommended to use a value less than or equal to the number of physical CPU cores.

2. **Typical Runtime:** The script can process files extracted from 100 chloroplast genomes in about 10 seconds using a 3-core CPU.

add_profile.py

1. Function Description

This script merges new sequence files into existing multiple sequence alignment results using MAFFT's profile alignment feature. It preserves the original sequence header information during the process.

2. Prerequisites

- **Python Environment:**
  - **Python 3.x is required.**
  - **Install the necessary libraries: biopython and argparse.**
  - **Installation command: pip install biopython argparse**
- **External Dependency:**
  - **The MAFFT alignment tool (refer to the official documentation).**
  - **The script automatically detects mafft or mafft.bat in your system's PATH. If it's not found, you will be prompted to enter the path manually.**

3. Parameter Description

Parameter	Abbreviation	Required	Default	Description
--input_dir	-i	Yes	-	Directory storing existing alignment results (in FASTA format)
--add_profile	-a	Yes	-	Directory of new sequences to be merged (in FASTA format)
--para_file	-p	No	3	Number of files processed in parallel (default value: %(default)s)
--thread	-t	No	1	Number of threads used by MAFFT for each task (default value: %(default)s)

Note： The files in the input directory and the directory to be merged must have the same names (e.g., gene1.fasta must exist in both directories).

4. Important Note

The files in the input directory and the directory to be merged must have the same names (e.g., gene1.fasta must exist in both directories).

5. Usage Examples

```
Bash
# Basic usage (auto-detects MAFFT path)
python 8add_profile.py -i aligned_results -a new_sequences

# Specify parallel parameters (4 parallel tasks, 2 threads per MAFFT task)
python 8add_profile.py -i base_aligns -a additional_seqs -p 4 -t 2
```

6. Processing Workflow

1. **Path and File Detection:** The script automatically searches for the MAFFT executable in your system's PATH. If it can't find it, it will prompt you for the correct path. It also verifies that the input and additional directories exist and are not empty.

2. **File Matching:** The script scans both directories for identically named FASTA files (with extensions .fasta, .fas, or .fa) and creates a task queue, where each task is a pair of matching files.

3. **Parallel Merging:** The script uses MAFFT's --reorder --add command to merge the new sequences into the existing alignment. It uses temporary files to ensure atomic operations, then overwrites the original alignment file upon successful completion.

7. Output

- **The merged results directly overwrite the original files in the --input_dir.**

- **The original sequence header information is preserved; only the sequence content is updated.**

8. Technical Details

- **MAFFT Parameters:**

    - --reorder: Keeps the output sequence order consistent with the input.
    - --add: This enables profile alignment, which adds new sequences to an existing alignment.

- **Error Handling:** The script includes checks for nonexistent/empty files, validates the MAFFT path, and handles temporary file exceptions.
- **Performance Optimization:** The script uses a two-level parallelization approach (parallel files and threads per MAFFT task) and stores temporary files in the same directory to reduce I/O overhead.

9. Typical Scenarios

- Adding new species to an existing phylogenetic analysis.
- Batch processing samples in a metagenomic study.
- Incrementally updating alignments in large-scale sequencing projects.

Tip: When processing many files, it is recommended to set the -p parameter to the number of physical CPU cores and the -t parameter to 2-4 (adjusting based on available memory).

coverage_and_identity.py

1. Function Description

This script analyzes multiple sequence alignment results to calculate the coverage and identity of each sequence relative to a reference sequence (the first sequence in the file). It then filters out low-quality sequences based on user-defined thresholds and generates a detailed statistical report.

2. Prerequisites

1. Python Environment:

- Python 3.x is required.
- Install the necessary libraries: biopython, pandas, and numpy.
- Installation command: pip install biopython pandas numpy

2. Input File Requirements:

- Input files must be in FASTA format (.fasta, .fas, or .fa).
- The first sequence in each file is used as the reference.
- Sequences must already be aligned and contain gap symbols (-).

3. Parameter Description

Parameter	Abbreviation	Required	Default	Description
--input_dir	-i	Yes	-	Input directory (stores aligned FASTA files)
--output_dir	-o	Yes	-	Output directory (saves filtering results and statistical reports)
--threshold	-t	No	0.75	Threshold for coverage/identity (between 0-1)
--para_file	-p	No	3	Number of files processed in parallel (default value: %(default)s)

note：The thresholds you set directly impact the strictness of the results. It's recommended to adjust them based on your data type. For instance, you could use a higher threshold like 0.9 for highly conserved genes and a lower one like 0.6 for more diverse genes.

3. Usage Examples

```
Bash
# Basic usage (uses the default threshold of 0.75)
python coverage_and_identity.py -i aligned_cds -o filtered_results

# Custom thresholds (coverage and identity >= 80%)
python coverage_and_identity.py -i alignments -o output -t 0.8

# Using 8 parallel processes
python coverage_and_identity.py -i input_dir -o output_dir -p 8
```

4. Explanation of Metrics

- **Coverage:**
  Coverage=fractextNumberofvalidbasesinthesequenceatnon−gappositionsinthereferencetextTotallengthofthereferencesequence
  - This measures the **completeness** of a sequence relative to the reference, ignoring the influence of gaps.
- **Identity:**
  Identity=fractextNumberofmatchingbasesbetweenthesequenceandthereferencetextNumberofvalidbasesinthesequenceatnon−gappositionsinthereference
  - This measures the **similarity** of a sequence to the reference. It only compares positions where both sequences have a base.

5. Output

- **Filtered FASTA Files:**
  - These files contain all the sequences that passed the threshold checks.
  - The filenames are the same as the input files (e.g., gene1.fasta becomes gene1.fasta).
- **Statistical Report (CSV Format):**
  - A separate report is generated for each input file.
  - The filename format is <original_filename>_stats.csv.
  - The report includes four columns of data.

Column	Description
description	Sequence description information
coverage	Coverage (4 decimal places)
identity	Consistency (4 decimal places
filter_res	Filtering result (PASS/FAIL)

6. Processing Workflow

1. Read Alignment File: The script reads a FASTA file, treating the very first sequence as the

reference. It calculates the effective length of the reference sequence by excluding any gap characters (-).

2. Calculate Metrics: The script iterates through each sequence in the file and compares it to the reference. It only considers positions where the reference has a valid base (not a gap), then counts the number of matching and mismatching bases for each sequence.

3. Filter Sequences: Sequences must meet a dual condition to be kept: their coverage must be greater than or equal to the threshold AND their identity must also be greater than or equal to the threshold. Sequences that fail either of these checks are discarded.

4. Output Results: The script generates a new FASTA file containing only the sequences that passed the filtering. It also creates a detailed CSV report with metrics for all sequences, including those that were discarded.

rename.py

1. Function Description

This script preprocesses FASTA files, offering two main functions:

- **Export Sequence Information:** Parses FASTA record headers based on a custom format, extracts key information, and generates a summary table.
- **Correct Biological Annotations:** Uses a correction table to batch-correct organism names within FASTA files.

2. Prerequisites

- **Python Environment:**
  - Python 3.x is required.
  - Install the necessary libraries: **biopython** and **pandas**.
  - Installation command: pip install biopython pandas
- **Input File Requirements:**
  - FASTA files must contain aligned sequences (all sequences must be of the same length).
  - Record headers must have a uniform format that adheres to a user-defined specification.
  - The correction mode requires an additional CSV correction table with accession and organism columns.

3. Parameter Descrption

Parameter	Abbreviation	Required	Default Value	Description
--input	-i	Yes	-	Input path (file or directory)
--output_dir	-o	Yes	-	Output directory
--para_file	-p	No	3	Number of files processed in parallel (only valid in directory mode)

4. Operation Modes

After running the script, you'll be prompted to select an operation:

```
Bash
Select operation:
1 - Export records information    # Export sequence information
2 - Fix organism annotations      # Correct organism annotations
0 - stop execution                # Stop execution
```

5. Usage Examples

```
Bash
# Export sequence information from all FASTA files in a directory
python 10rename.py -i ./fasta_dir -o ./output -p 4
```

```
# Select option 1, and provide the header format string, e.g., "accession:l
ocation|organism|description"

# Correct organism names in a single FASTA file
python 10rename.py -i input.fasta -o ./corrected
# Select option 2, provide the header format, and the path to a CSV correct
ion table
```

6. Format String Explained

The format string defines the rules for parsing the FASTA record headers.

- Field Names: Prehttps://www.google.com/search?q=-defined fields include:
    - accession (required)
    - location (required)
    - organism (required)
    - description (optional)
    - other information (optional)
- Delimiters: Any combination of characters.

Example:

```
Python
"accession:location|organism|description| other information "
This string would parse a header like >NC_12345: join{[83632:84023](-), [82
533:82967] |Arabidopsis_thaliana|Chloroplast|Sample other information .
# accession: "NC_12345"
# location: " join{83632:84023, [82533:82967] "
# organism: "Arabidopsis thaliana"
```

7. Processing Workflow

1. Export Records Information Mode:

- Parses the headers of all FASTA files.

- Extracts the accession and organism fields.
- Generates a CSV summary table (seq_info.csv) with unique entries.
- Performs a strict check for duplicate accession and organism values, which must be unique.

2. Fix Organism Annotations Mode:

- Loads a CSV correction table (which must contain accession and organism columns).
- Matches original records using the accession numbers.
- Generates a new header format: >{accession}:{location}|{organism}.
- Outputs the corrected FASTA files, keeping the original filenames.

8. Output

- **Export Mode:**
  - **seq_info.csv:** A CSV file with two columns.

列名	说明
accession	序列编号
organism	生物体名称

- **Correction Mode:**
  - FASTA files with the same name as the input, but with a standardized header format:

```
Fasta
>{accession}:{location}|{organism}
```

9. Important Notes

- **Key Checks:**
  - **Directory Mode:** The script automatically verifies that all FASTA files are aligned (i.e., all sequences have the same length).

    - **Export Mode:** It ensures that there are no duplicate values in the accession or organism fields.
    - **Correction Mode:** The CSV correction table must cover all accession numbers in the input files.

- **Error Handling:**
    - The script will stop running if it finds duplicate accession numbers or organism names.
    - Detailed error messages are displayed if header parsing fails.
    - It will also throw an error if the format string is missing a required field

assemble_with.py

1. Function Description

This script merges two gene sequence datasets from different sources (a CDS sequence dataset and a short gene fragment dataset). It selects the optimal sequence based on a specified mode and constructs an integrated gene matrix.

2. Prerequisites

- **Python Environment:**
  - Python 3.x is required.
  - Install the necessary libraries: biopython, pandas, and numpy.
  - Installation command: pip install biopython pandas numpy
- **Input File Requirements:**
  - FASTA files must have a uniform sequence header
    format: >{accession}:{organism}|...
  - Each FASTA file should represent the sequences of all species for a single gene.
  - Within a single gene file, each species can only have one sequence.

3. Parameter Description

Parameter	Abbreviation	Required	Description
--input_CDS_dir	-c	Yes	CDS sequence directory (.fasta files)
--input_other_gene_dir	-g	Yes	Other gene sequence directory (.fasta files)
--output_dir	-o	Yes	Integrated result output directory
--mode	-m	Yes	Sequence selection mode (1/2/3)

Sequence merging mode description

Mode	Selection Strategy	Applicable Scenarios
1	Prioritize retaining CDS sequences	When CDS data quality is higher

2	Prioritize retaining small fragment gene sequences	When small fragment gene data is more complete
3	Retain the longest sequence	When maximizing sequence information is needed

4. Usage Examples

```
Bash
# Mode 1: Prioritize CDS sequences
python assemble_with.py -c cds_sequences -g other_genes -o merged_output -m
 1

# Mode 3: Keep the longest sequence
python assemble_with.py -c chloroplast_genes -g nuclear_genes -o combined -
m 3
```

5. Processing Workflow

1. **Input Validation: The script first checks if both the CDS and other gene directories exist and then creates an output directory if it doesn't.**
2. **File Categorization: It categorizes the files into three groups:**
    - **Files unique to the CDS directory.**
    - **Files unique to the other genes directory.**
    - **Files that are present in both directories.**
3. **Sequence Merging Strategy:**
    - **Unique files: These are directly copied to the output directory.**
    - **Common files: The merging strategy is determined by the selected mode:**
        - **Mode 1: The CDS sequence overwrites the other gene sequence.**
        - **Mode 2: The other gene sequence overwrites the CDS sequence.**
        - **Mode 3: The script selects the longest sequence for the same species.**
4. **Sequence Selection Algorithm:**

```
Python
if mode == 1:
    combined = pd.concat([df1, df2]) # CDS priority
    result = combined.drop_duplicates('organism', keep='first')
elif mode == 3:
```

```
    combined = pd.concat([df1, df2])
    result = combined.loc[combined.groupby('organism')['length'].idxmax()]
# Length priority
```

6. Output

- The output directory contains all the integrated gene FASTA files.
- Filenames are the same as the input files.
- Within each gene file, species names are unique.
- Sequence header format: >{accession}|{organism}|...
- Sequence content is the result of the optimal selection.

7. Important Notes

- **Key Requirements:**
  - **Within each gene file, each species must be unique (the script handles duplicates automatically).**
  - **Sequence headers must contain the accession and organism fields.**
  - **All sequences must be pre-aligned and have the same length.**
- **Typical Applications:**
  - **Integrating chloroplast and nuclear genome data.**
  - **Merging sequences from different sequencing platforms.**
  - **Combining public database data with experimental data.**
- **Performance Optimization: The script uses Pandas for efficient data manipulation and parallel processing for each gene file. It also uses a memory-friendly streaming approach.**

Tip: The mode you select directly impacts the quality of the output. Choose a mode that best suits your data:

- Mode 1 (CDS first) is ideal for conserved genes.
- Mode 3 (longest first) is best for highly variable genes.
- Mode 2 (other genes first) can be used to fill in missing data.

supermatrix_creator.py

1. Function Description

This script merges aligned sequences from multiple genes into a supermatrix for phylogenetic analysis. It also generates a partition file, which is essential for building large-scale phylogenetic trees.

2. Prerequisites

- **Python Environment:**
  - Python 3.x is required.
  - Install the necessary library: biopython.
  - Installation command: pip install biopython
- **Input File Requirements:**
  - The input files must be multiple sequence alignment files in FASTA format (and already aligned).
  - The sequence header format must be >{any_ID}|other_info...|{species_name}, where the species name is the last piece of information after the final |.
  - Within each gene file, there should only be one sequence per species.

3. Parameter Description

Parameter	Abbreviation	Required	Default	Description
--work_dir	-w	Yes	-	Working directory (contains all input FASTA files)
--output_phy	-o	No	"supermatrix.phy"	Output PHYLIP filename
--partition_file	-pf	No	"partitionFile.txt"	Output partition filename
--patterns	-p	No	["*.fasta"]	File matching patterns (supports wildcards)

4. Usage Examples

```Bash
# Basic usage
python supermatrix_creator.py -w ./alignments -o supermatrix.phy -pf partit
ions.txt

# Custom file patterns
python supermatrix_creator.py -w gene_data -p "*.fas" "*.fasta" -o combine
d.phy
```

5. Output

- **PHYLIP-formatted Supermatrix:**
    - The first line contains the total number of species and the total number of sites.
    - Subsequent lines contain the species name (left-aligned to 10 characters) followed by the concatenated sequence.
    - Example:

```Plaintext
3 15
Species1   ATGCTAGCTAGCTAG
Species2   ATG---GCTAGCTAG
Species3   ATGCTAGCTAG----
```

- **Partition File:**
    - This file records the position range of each gene within the supermatrix.
    - Format: {gene_name} = {start_position}-{end_position}.
    - Example:

```Plaintext
gene1 = 1-300
gene2 = 301-650
```

6. Processing Workflow

1. **Preparation:** The script switches to the specified working directory, scans for all FASTA files that match the given patterns, and validates the integrity of these files.

2. **Data Collection:** It extracts all species names based on the sequence header format, records the length of each gene, and calculates the total number of sites.

3. **Sequence Concatenation:** An empty sequence of the full length is created for each species. The script then fills in the valid sequences in gene order, padding any missing data with gap characters (-).

4. **Output Generation:** The script writes the PHYLIP-formatted supermatrix and generates the partition file, which documents the position of each gene.

7. Key Features

- **Intelligent Species Handling:** The script automatically identifies the species name from the last segment of the sequence header (after the final |) and handles cases where species are missing from certain genes.

- **Comprehensive Statistics:** It displays a list of processed gene files, reports the length of each gene, and calculates the total number of sites and species.

- **Error Handling:** Invalid files are skipped, and the script displays a warning. It also checks for empty files and validates file paths.

8. Important Notes

- **Sequence Header Format: The header must include a | separator, and the species name must be in the final part of the header. For example: >GeneID123|Other info|Arabidopsis_thaliana.**

- **File Requirements:**

    - All input files must be aligned (i.e., sequences must have the same length).

    - Species names must be unique within a single gene file.

    - File extensions must match the specified patterns.

- **Output Format: The PHYLIP format is compatible with popular phylogenetic software such as RAxML and IQ-TREE.**